Real-time generalization of point data in mobile and web mapping using quadtrees

Bereuter, Pia ; Weibel, Robert

Abstract: With a focus on mobile and web mapping, we propose several algorithms for on-the-fly generalization of point data, such as points of interest (POIs) or large point collections. In order to achieve real-time performance we use a quadtree data structure. With their hierarchical subdivision structure and progressive levels of detail, indices of the quadtree family lend themselves as auxiliary data structures to support algorithms for generalization operations, including selection, simplification, aggregation, and displacement of point data. The spatial index can further be used to generate several local and global measures that can then serve to make educated guesses on the density and proximity of points across map scales, and thus enable control of the operation of the generalization algorithms. An implementation of the proposed algorithms has shown that thanks to the quadtree index, real-time performance can be achieved even for large point sets. Furthermore, the quadtree data structure can be extended into a caching structure, which can be used to store pre-computed generalizations; thus, a desired level of detail can simply be retrieved from cache.

DOI: https://doi.org/10.1080/15230406.2013.779779

Real-time Generalization of Point Data in Mobile and Web Mapping Using Quadtrees

Pia Bereuter* and **Robert Weibel**
Department of Geography
University of Zurich
Winterthurerstrasse 190
8057 Zurich
Switzerland
Email: {pia.bereuter | robert.weibel}@geo.uzh.ch

ABSTRACT: With a focus on mobile and web mapping, we propose several algorithms for on-the-fly generalization of point data, such as points of interest (POIs) or large point collections. In order to achieve real-time performance we use a quadtree data structure. With their hierarchical subdivision structure and progressive levels of detail, indices of the quadtree family lend themselves as auxiliary data structures to support algorithms for generalization operations, including selection, simplification, aggregation, and displacement of point data. The spatial index can further be used to generate several local and global measures that can then serve to make educated guesses on the density and proximity of points across map scales, and thus enable control of the operation of the generalization algorithms. An implementation of the proposed algorithms has shown that thanks to the quadtree index, real-time performance can be achieved even for large point sets. Furthermore, the quadtree data structure can be extended into a caching structure, which can be used to store pre-computed generalizations; thus, a desired level of detail can simply be retrieved from cache.

KEYWORDS: mobile and web mapping, on-the-fly generalization, real-time generalization, point data generalization, POI, auxiliary data structure, quadtree

Introduction

In comparison to map generalization for paper maps, map generalization for mobile mapping and interactive web mapping has a different set of requirements, in particular its need for on-the-fly (or real-time) generalization and adaptation to user interaction and content. Web or mobile mapping applications, such as mashups or location-based services (LBS), typically display some thematic foreground data — predominantly in the form of point data such as points of interest (POIs) or large point collections — against the spatial reference provided by some background data (e.g. a topographic map or orthoimagery). A POI refers to a point location that carries a particular meaning that might be of interest to the map user, such as restaurants, sports facilities, civic infrastructure etc. (an 'amenity' in OpenStreetMap). The background data is usually assumed to be static in content and can thus be rendered by a pre-generalized tile service (e.g. map ser-

vices based on OpenStreetMap, Google Maps, Bing Maps) to provide seamless map inter action. The foreground data, on the other hand, will vary in content depending on the user's request. For example, the POIs displayed as foreground will only encompass restaurants, or even only a certain type of restaurant, if finding a restaurant is the task of the user; and the foreground may change to parking lots in the next request, when the user then wants to find a suitable place to park near the chosen restaurant. Thus, cartographic generalization of foreground data has to be achieved in real-time, requiring flexible, on-the-fly generalization algorithms (Weibel and Burghardt, 2008). However, so far only few algorithms for on-the-fly generalization of point data have been proposed in the literature. As a quick survey of online mapping applications will easily show, most of these applications simply render the foreground point data in an ungeneralized way, potentially resulting in massive overlaps and clutter, depending on the density and distribution of the point data. Such applications are usually implemented as mash-ups on the basis of map services such as the ones mentioned above. Of course, since the map service is interactive, the user has the option to zoom in further until spatial conflicts among the foreground data disappear. On the other hand, the overview of the overall spatial distribution will be lost.

Thus, we believe that on-the-fly generalization of large point data sets is necessary in web and mobile map services. Such generalization functions should be sufficiently fast to achieve real-time performance, and they should be flexible enough to adapt to varying user requirements and contexts of usage.

The contribution of this paper consists of a set of algorithms that use a quadtree index as an auxiliary data structure. Using this index and search structure allows achieving real-time performance for large point sets. The quadtree also supports the computation of various measures such as local point density, and allows the implementation of different generalization operators, including selection, simplification, aggregation, and point feature displacement. Using these different generalization operators informed by measures computed from the quadtree, mobile mapping applications for point data can achieve flexibility and adaptation. The proposed generalization algorithms and the creation of the quadtree data structure were both performed on-the-fly on a prototype map client. The data was not assumed to be known *a priori* (cf. the above stated requirement of changing the foreground point data). However, it is important to note that the algorithms are generic and not restricted to be performed solely on a map client. Architectural considerations of client vs. server-side computing are not the issue of this paper.

Background and Related Work

Generalization Operators

In map generalization different generalization operators are applied to reduce spatial conflicts (McMaster and Shea, 1992). Due to the fact that in most maps geographic space is not distorted, object-directed approaches are mostly used (Bereuter and Weibel, 2010), where map objects not map space are modified. In the case of point features four object-directed generalization operators (Figure 1) are of interest: *selection, simplification* and *aggregation* reduce the number of features represented on the map, while *displacement* moves the map features away from each other to remove overlaps and congestion.

Among the point reduction operators, selection and simplification choose a subset of the original points, while aggregation typically generates new point positions (placeholders). The difference between selection and simplification is the same as in the classification by McMaster and Shea (1992): selection is based on attributes, while simplification is based on geometric criteria. Note that generalization *operators* define a particular generalization process in a conceptual way; they are implemented by generalization *algorithms*. For a given operator, several algorithms are usually possible (McMaster and Shea, 1992).

Each of the above generalization operators has different requirements. However, for all operators the detection of spatial conflicts is crucial. For selection and aggregation, hierarchical ordering plays a key role, while displacement necessitates region and neighborhood queries. A hierarchical data structure like the quadtree has both, hierarchical spatial order and the facility to speed up region and neighborhood queries.

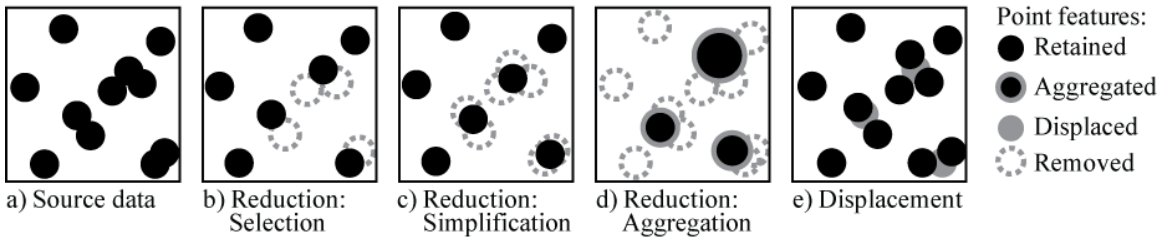


Figure 1 Overview of point generalization operators.

Related Work

On-the-fly generalization for mobile mapping is typically achieved by relying either on fast generalization algorithms or on pre-computation and hierarchical data structures (van Oosterom 2005; van Oosterom and Meijers, 2011; Weibel and Burghardt, 2008). The former approach commonly has to sacrifice cartographic quality to reduce computational complexity and achieve speed, while the latter, as a consequence of pre-computation, lacks flexibility.

Solutions of the first group — fast generalization algorithms — typically rely on rather simple but effective algorithms and heuristics. The preference is on rather straightforward generalization operators such as selection and simplification. For instance, feature selection based on the Radical Law (Töpfer and Pillewizer, 1966) and ordered attributes (Lehto and Sarjakoski 2005), or applying local priority criteria (Edwardes, et al. 2005).

The second group represents methods that fully rely on pre-computed generalizations stored in hierarchical data structures. Van Oosterom (2005) and van Oosterom and Meijers (2011) review and propose several data structures for real-time generalization. Initial ideas to use quadtrees and hierarchical drainage basins to support generalization have been proposed by Burghardt et al. (2004) and Edwardes et al. (2005). Dutton (1999a, 1999b) developed a space-efficient, quadtree-like encoding scheme for positions on the globe called the quaternary triangular mesh (QTM), and used it for line filtering in map generalization . De Berg et al. (2004) describe an algorithm using a k-d-tree.

What is still missing are methods combining the advantages of both approaches, that is, real-time algorithms for point generalization exploiting hierarchical spatial data structures. This research gap provides the point of departure for the algorithms proposed here.

Properties of Quadtrees

The quadtree is a well known spatial tree data structure and a generic term for a family of tessellations of the plane in which every node has four children (Samet, 1989). It is widely used for 2-D spatial indexing in GIS or other domains, such as in computer graphics for collision detection (Moore and Wilhelms, 1988). Since our case is point data generalization the point region (PR) quadtree was selected, as it has several properties that make it useful for real-time generalization. For reviews of quadtrees and associated algorithms, see Samet (1984) or Samet (1990).

Properties of quadtrees that are useful for real-time generalization are listed in Table 1. The spatial index speeds up spatial queries and search. The spatial coverage of quadtree tiles provides information on existence or absence of geographic features in a specified region and thus enables estimates on feature density/distribution. The topology of quad neighbors provides information about the local neighborhood structure (Samet 1989). And the recursive hierarchical subdivision of map space adapts to point density and thus progressively builds up a spatial hierarchy with implicit scale progression.

Table 1: Properties of quadtrees useful for real-time point generalization

Property	*Use*	*Generalization operator*
Spatial Index	Speed up spatial search	Selection, aggregation
Coverage	Enables estimates on densities and distribution	Selection, displacement
Topology	Quad neighborhood	Displacement
Hierarchy	Recursive and progressive subdivision	All operators

Quadtree-Based Algorithms for Point Data Generalization

General Overview

The basic idea of the quadtree-based generalization approach is to apply generalization operations to quadtree nodes according to the target *level of detail* LOD, which is mapped to the depth of a quadtree. The basic operations on quadtrees include: *insert*, *delete*, *get neighbors*, and *query* the quadtree (Samet 1989, 1990). They form the foundation of the generalization algorithms described below. The shape of the tree depends, and therefore reflects, the spatial distribution of the inserted point set and is independent of the insertion order. The quadtree is built in real-time after the point set is either loaded from a local data repository or via a spatial query to a server (e.g. select restaurants within the greater Zurich area).

The generic flow of our quadtree-based generalization approach consists of three steps:

1. Creation: In a first step the quadtree is created in real-time by inserting the loaded point data and their (optional) attributes into the data structure, covering the extent of the query window, and projecting the data from lat/lon coordinates to the coordinate system of the tiling service. Attributes such as feature category, rank (e.g. relevance ranking generated by an external application), or other measures are assigned to the point data if required by the desired generalization algorithm. In our implementation, attributes are stored externally to the quadtree and accessed via the point objects' unique identifiers.

2. Generalization algorithm: In the second step, a generalization algorithm returns for each quadnode the resulting generalized points at the *target LOD*. The target LOD (and thus target map scale) is mapped to the tree depth. It translates to the width of the quadnode side, measured in screen (pixel) coordinates (Fig. 2d) and denotes the smallest required distance to resolve spatial conflicts, given cartographic constraints (e.g. point symbol size). In other words, the target LOD is defined such that the symbol width is smaller than the width of the quadnode at the target LOD. For each visited quadnode at the desired depth either a generalization algorithm is applied in real-time to all its childnodes, or the generalized points are retrieved from previous, pre-computed runs (see "Caching"). In order to fully meet cartographic constraints, point symbols overlapping the border of quadnodes may be displaced if a check of quadnode neighbors reveals that there is an overlap with another point. For details of generalization algorithms, see the following Subsection.

3. Display and caching: In the third step the results from Step 2 are either displayed (display-and-forget), or *optionally* stored in the nodes for fast retrieval in subsequent iterative generalization (see "Caching").

Quadtree-Based Generalization Algorithms

This section presents different quadtree-based generalization algorithms for the generalization operators of Figure 1. The proposed algorithms consist of those that, for a given quadnode, derive generalization results based solely on the quadnode and its subtree, and those that consider also neighboring nodes. The second group is computationally more expensive but leads to cartographically superior results. First, however, we briefly review the role of geometrical measures in support of quadtree-based generalization algorithms.

Measures. Geometrical measures help to inform the operation of generalization algorithms, and parameterize their outcome (Bereuter and Weibel, 2011). *Local measures* are derived from each quadnode and its neighbor nodes, such as the number of points stored in the subtree, maximum depth, size, and balancing of the subtree. *Global measures* are based on the complete dataset and include global statistics such as the average number of elements per node and LOD. Such measures can be used to control the generalization and portrayal process. For instance, the number of points stored in the subtree of the current quadnode at the target LOD can be directly used to adapt the point symbol size in aggregation algorithms and yield graduated symbols (as shown in Fig. 9). Measures can also be used to set thresholds for the generalization algorithm. For instance, when pruning elements from the tree preference can be given to quadnodes that represent a large subtree, and thus a large number of points, at target LOD. Thus, the underlying spatial pattern can

be retained. The use of measures as parameters in the proposed algorithms thus helps to maintain the balance between local and global maxima of the overall spatial distribution of the data.

Selection. Chooses a subset of points from the original set of points, based solely on attributes, such as a relevance value per point object (Bereuter and Weibel, 2010).

Value-based selection and *rank-based selection*. Returns the most significant element per quadnode (Fig. 2b). 'Most significant' denotes a function of a numeric attribute, such as maximum value of an attribute of points occurring in a subtree. For POI data, such as restaurants (cf. Fig. 6a), highest relevance may be used (obtained from a relevance ranking service). For point collections (i.e. point data sets that encompass large collections of counts or observation data), such as animal observation data (cf. Fig. 6b), the most or least frequently occurring observation may be used, or a weight to maintain the overall distribution of counts to avoid over or under-representation of certain feature categories (e.g. ibex in Fig. 6b). If the numeric attribute of the point is normalized by the depth or the total number of points of the quadnode's subtree, the algorithm will generate a generalization result that approximates the local point density.

Simplification. Like selection, it chooses a subset of the original points. However, it is governed by geometric properties, as in line simplification (Bereuter and Weibel, 2010; McMaster and Shea, 1992).

Centrality-based simplification. Retains the most *central point* per quadnode, defined by the point lying closest to the mean center of the points in a quadnode (black dots in Fig. 2c). Thus, solely requires the positions of the point features.

Weighted centrality-based simplification. Either applies the weighted mean by using a numeric attribute to derive the most central point per quadnode, or the central point is obtained considering also the points of the neighboring quadnodes, potentially leading to a more balanced generalization result. Both variants are shown schematically in Figure 2d.

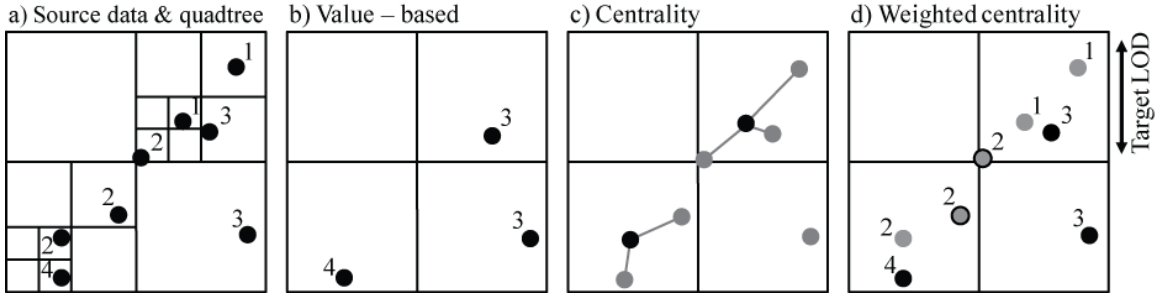


Figure 2 Point reduction algorithms, with results for depth 1: a) quadtree source data; b) value-based selection, retaining points with highest attribute value per subtree; c) centrality-based simplification, retaining points closest to the mean center of all points contained in a quadnode. d) two variants for weighted centrality simplification: central point obtained by weighted mean of attribute values within quadnode only (black dots), vs. central points obtained considering also the neighboring quadnodes (dots with black outline).

Aggregation. Reduces the number of points by grouping together semantically similar or spatially close points (Bereuter and Weibel, 2010), replacing the original points by a new placeholder feature (i.e. the point positions change).

Quadnode center aggregation. Aggregates points of a quadnode to the position of the corresponding tile center. This is the most basic of the proposed algorithms.

Midpoint aggregation. Aggregates to the midpoint (i.e. mean center) of the points of a quadnode's subtree (Fig. 3b). Uses solely the positions of the points. As in weighted centrality simplification, neighboring quadnodes may be taken into consideration, e.g. by weighting the mean center by the number of points stored in the neighboring quadnodes.

Cluster-based aggregation. Returns a placeholder (e.g. the modal center) for highly clustered and densely populated quadnodes with a large number of children, based on the positions and attributes of the points (Fig. 3c). Assigns the points to the same cluster if the neighboring childnodes (four respectively eight connected neighbors) are not empty or yield a minimum, user-defined number of points.

Collocation filtering. Generalizes quadnodes based on a collocation rule, such as the co-occurrence of features in a quadnode belonging to the same class, or capturing logical relationships between different point categories (Fig 3d). Collocation of one or more point features in space seems to influence the user's decision in location-based services (Reichenbacher and De Sabbata, 2011). An example for collocation is finding collocated parking lots and restaurants in a city. For each quadnode's subtree the collocation algorithm checks if it contains restaurants *and* parking lots, and if so, how often, in order to scale the resulting point symbols proportionally to the number of co-occurrences of restaurants and parking spaces within that node resulting in one aggregated symbol (Fig. 3d) or two separate symbols (Fig. 9b) per generalized quadnode. To avoid edge effects not considering collocated elements across boundaries of two neighboring quadnodes the algorithm can be extended to check for collocation also in neighboring nodes. Note that while collocation filtering conceptually might be thought to be a selection operation, we count it among the aggregation algorithms, since we aggregate multiple co-occurrences within a quadnode and its subtree.

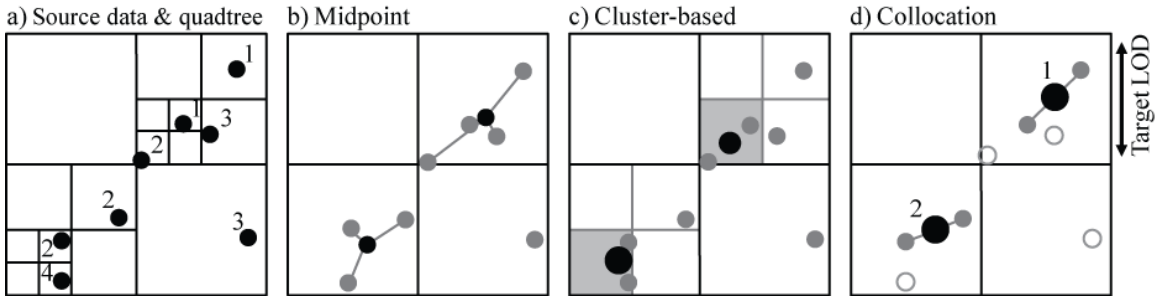


Figure 3 Aggregation algorithms: a) quadtree source data, b) midpoint aggregation; c) cluster-based aggregation; d) collocation filtering for co-occurring same valued points.

Displacement. Locally reconfigures point symbols to resolve spatial conflicts by moving points represented. It is thus usually applied for the 'finishing touches', as the last in the chain of generalization operators. Its application is limited to resolve spatial conflicts in a narrow band of scales, typically from one LOD to the subsequent one.

Our quadtree-based displacement algorithm acts – as the previously presented quadtree algorithms – on the target LOD. The algorithm reallocates points of a quadnode not satisfying cartographic proximity constraints (framed red in Fig. 4) to neighboring quadnodes.

Points can be displaced to their cardinal neighbors depending on whether the king's or rook's move is applied (Fig. 4a, b). Therefore for each point stored inside the four childnodes there are two (rook's case) or three (king's case) favorable displacement options. The actual *direction* in which a point is displaced is preferably directly opposed to the direction of the mean center formed by all the quadnodes points (Fig. 4e). The preferred range of directions in which a point can be displaced is limited by the bisectors of its immediate neighboring points to the mean center and by the holding capacity of the neighbor nodes.

A point can only be displaced to the neighboring quadnode if that node shares the same depth and is empty, or has a lower depth (grey shaded quadnodes in Fig. 4d). This is controlled by the quadnodes' *holding capacity*: it indicates how many points the neighboring quadnode can possibly contain, without further subdivision (Fig. 4d). For each point the holding capacity for each possible displacement direction is summed up and the points are sorted in ascending order (Fig. 4c). The point closest to the midpoint of the points is set to the end of the list. The childnode with the least possibilities for displacement is displaced first, reducing that neighbor's holding capacity by one. If there is no other further possibility of displacement the point is kept. If the neighboring quadnode is of a depth less than the current quadnode and already holds one point, this node will be further subdivided to insert the moved point (Point C in Fig. 4f). The subdivsion, however, will not exceed the target LOD. If after subdivision both points happen to be inside the same childnode, the moved point is discarded. The goal is, at the limit of display resolution (i.e. at the target LOD), to have one or zero points per node.

If no more points can be moved the algorithm stops and keeps the most central point (Point A in Fig. 4f) or the one with the highest importance value and removes the childnodes and remaining points that could not be displaced.

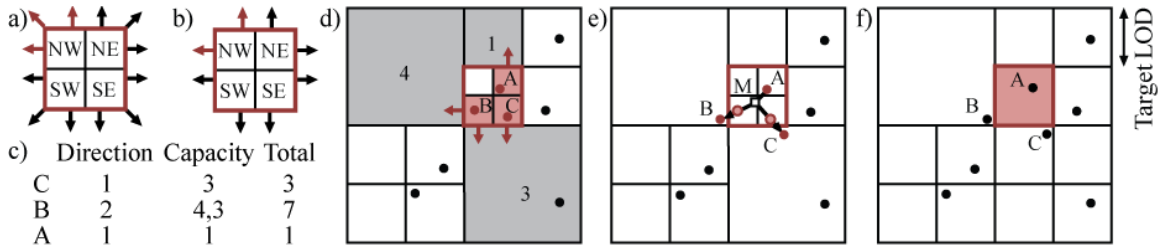


Figure 4 Steps applied by the displacement algorithm for the red quadnode: a, b) movement options per child quadnode (rook's move vs. king's move). c) sorted table of summed holding capacity per point. d) Points A, B, C stored in the childnodes are candidates for displacement. Gray shaded areas denote candidate target quadnodes for displacement, numerals denote their holding capacity. Red arrows indicate the possible movement direction for the points contained in childnodes. e) Optimal displacement directions for B and C. A is kept, as it is the most central point. f) B and C are displaced and the childnodes removed.

Extension. The above algorithm can be extended by allowing point displacement to be propagated to neighbors of degree 2. If a point cannot be displaced to its immediate (degree 1) neighbors using the above algorithm, the algorithm would try to displace an immediate neighbor in the preferred direction of displacement first, to make room for displacement. For performance reasons, it is advisable to limit the search to a pre-defined, small degree of neighborhood. Due to lack of space, the extended algorithm will not be further elaborated on.

Caching

The above algorithms are designed for real-time performance and no caching (i.e. temporary storage) is needed (display-and-forget mode of operation). However, the quadtree also lends itself nicely as a caching structure for generalization results (rather than serving merely as spatial index). Caching of the generalization results directly in the nodes of the quadtree facilitates faster interactive zooming, as long as the generalization algorithm and/or the search criteria do not change. If the application requires frequent change of generalisation algorithms, the result may be stored using identifiers for the respective generalisation result stored in the quadnodes.

Figure 5 schematically illustrates the use of the quadtree as a caching structure. In display-and-forget mode (i.e. the normal case), the generalization algorithm returns the results for each visited node and its subtree, but does not store it persistently (Fig. 5a shows an example for LOD 1). However, in the same pass, the results of the generalization algorithm can also be stored in each quadnode visited. If results have been cached and if, due to zooming, the same LOD is requested a second time, the portrayal process can simply retrieve the cached result from each quadnode at the requested LOD, optimizing the portrayal process in terms of speed. Figures 5b and 5c show two different ways of exploiting the caching tree for retrieval. Figure 5b depicts the case when zooming out: for an LOD *k* that is visited in the ascending order, only the previous LOD *k*+1 needs to be retrieved to derive the full content of the LOD *k*. Figure 5c depicts the usage case typical of generating approximate generalization results. The search range per quadnode is restricted to a user-defined depth *k*+*i*, which depends on the generalization algorithm applied. For instance, for a displacement algorithm it is sufficient to only consider few LODs (e.g. LOD *k*+1), as displacement is an operation that acts only locally.

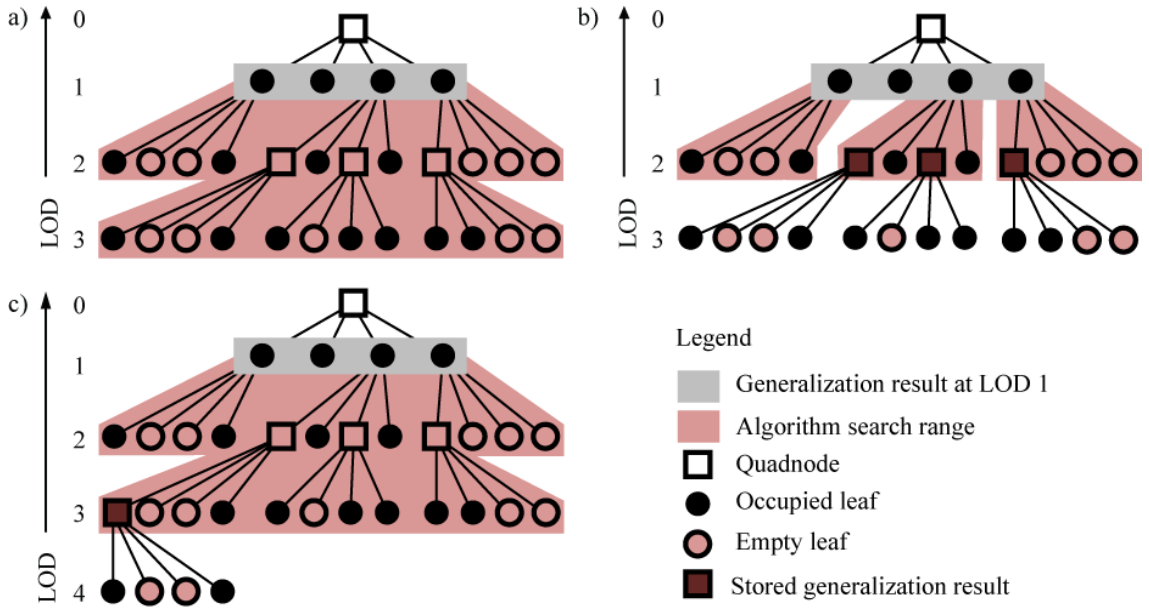


Figure 5 Retrieval from cache for generalization at LOD 1: a) for each quadnode its complete subtree is considered (equivalent to display-and-forget mode, i.e. no caching); b) for each quadnode at LOD k (= 1) its leaves and the generalization results (black squares with red fill) of the LOD k+1 (= 2) are considered; c) depth restricted retrieval with LOD k+i; all leaves up to k+i (= 3) and the generalization results of k+i (= 3) are considered.

Experiments

Prototype and Data

The proposed quadtree-based generalization algorithms have been implemented in a prototype generalization platform using Java and Processing (www.processing.org). Two datasets were used for the foreground data: a POI dataset of eating places in the Canton of Zurich, Switzerland originating from OpenStreetMap (Fig. 6a); and a point collection of cumulative animal observation counts provided by the Swiss National Park (Fig. 6b).

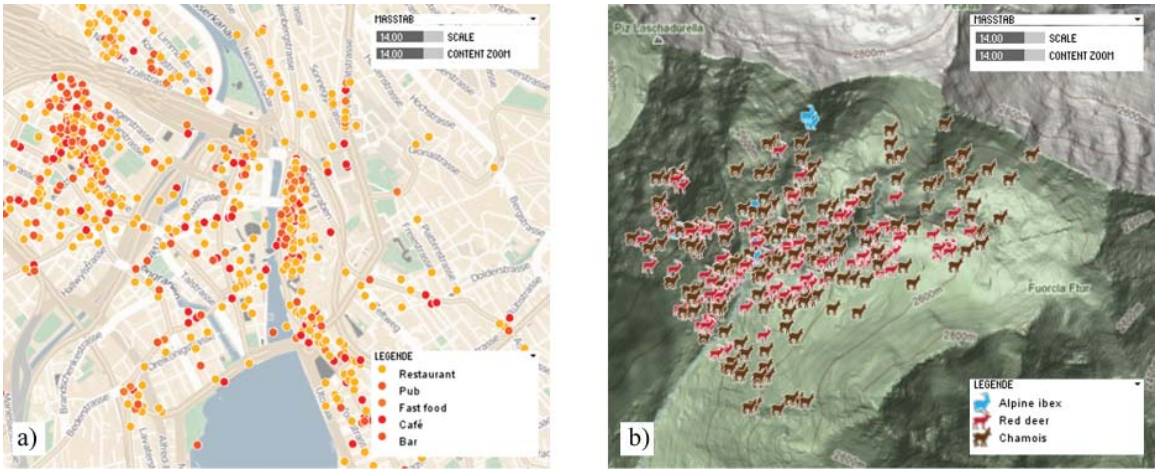


Figure 6 a) OSM dataset POI in Zurich, Switzerland depicting eating places by type. b) Cumulative animal observation data depicting animal types, courtesy Swiss National Park. Base map: a) © 2012 CloudMade – map data CC BY SA 2012 OpenStreetMap.org, b) © 2012 Google

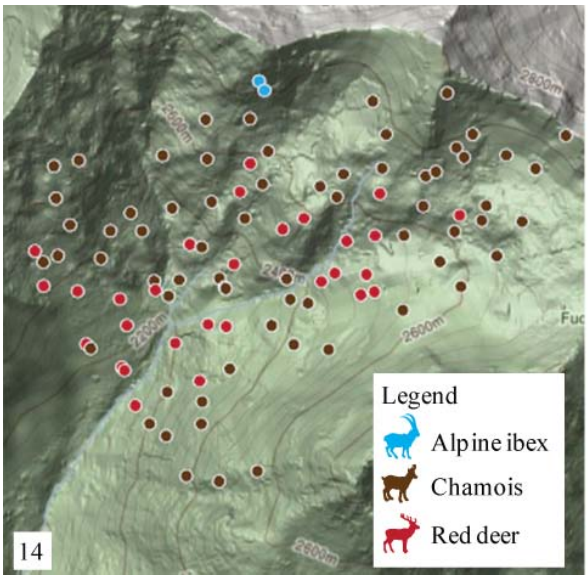
The POI dataset features several categories, from which a subset (restaurants, bars, cafés etc.) was extracted as a thematic layer for this paper. The animal data are cumulative, that is, the points represent observation counts at point locations, ranging from individual animals to entire flocks. While the point density varies for both data sets, the POIs of eating places exhibit a more marked, clustered density variation than the animal observations. For the background map of the POI data a generated soft-toned map style from CloudMade (www.cloudmade.com) was used. For the animal observation counts a shaded relief from Google Maps was used.

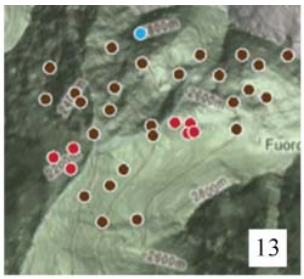
Results and Discussion

This section presents results generated by some of the quadtree-based generalization algorithms described above. Due to restrictions of space, we only show a portion of the study covered by the two data sets, and we only show results for a small selection of generalization algorithms.

Selection based on ranking or a selection function based on attributes only retains one representing element per quadnode. In Figure 7 only the point representing the highest animal observation count per quadnode is kept for a given LOD. Due to the strong selection confined to quadtree tiles, most of the spatial conflicts are resolved, though no displacement to resolve remaining overlaps was applied. The progression through LODs 14 to 12 shows how the generalization retains the spatial point configuration relatively well. Concerning the number of points retained, our algorithm differs quite strongly from Rad-

ical Law (Töpfer and Pillewizer, 1966). However, the Radical Law was originally developed for paper maps with more fine-grained map symbols, and thus has a tendency to retain too many points on screen maps where map symbols are invariably larger. The optimal number of points might thus be higher than the quadtree-based solution, but never reach the number defined by the Radical Law.





LOD	Scale	Points	Radical Law
14	~1:36000	98	98
13	~1:72000	34	69
12	~1:144000	10	49

Figure 7 Rank-based selection for LOD 14, 13, 12. For each quadnode, the algorithm retains the point with the highest observation count. Base map: © 2012 Google

While value-based selection retains values a user is most interested in, simplification by centrality keeps the most central point per quadnode (Fig. 8), that is, the point closest to the mean center of all points falling within the generalized quadnode. Figure 8b illustrates how the most central point is selected. As becomes noticeable, the selection of the most central point becomes more stable when more points are contained in a quadnode, while often generating questionable results for quadnodes containing only two points. Furthermore, clusters that consist of many points but happen to fall into the same quadnode are reduced to a single point, while single points that happen to fall into neighboring quads will persist as a group of four. This could be alleviated by additional checks for this special configuration.

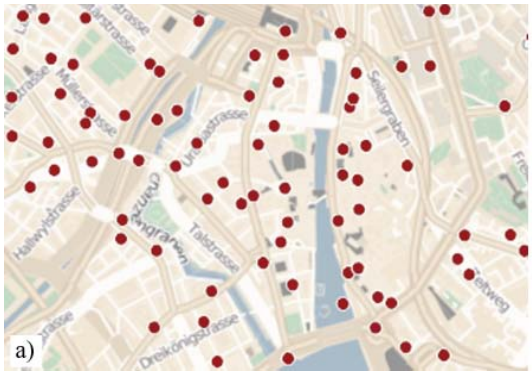


Figure 8 a) centrality-based simplification for LOD 14; b) enlarged debugging view: rectangles depict quadnodes, gray dots removed points, the gray square symbols denote the mean center of the points inside each quadnode, and the circle shows the distance to the closest point from the mean center. Base map: © 2012 CloudMade – map data CC BY SA 2012 OpenStreetMap.org

Aggregation groups two or more points and replaces them by a new placeholder point feature, with the point symbols optionally scaled by the number of points aggregated to generate a graduated symbol map. Note that proportional symbol size can be used as an additional cartographic option with all algorithms proposed above (not just aggregation), thus adding further information about the density of underlying original points. Figure 9a shows the result of midpoint aggregation, as blue, graduated circles proportional in size to the number of aggregated points. For comparison, the points resulting from ranking-based selection for the same LOD are overlaid. As can be seen, aggregation and selection are equivalent in areas of low point density, where quadnodes at the target LOD only contain a single point.

Collocation filtering is another type of aggregation where spatially co-occurring points are retained. Again, the map symbols depicting the resulting points may be weighted by the frequency of co-occurrence. Figure 9b side shows aggregation by collocation filtering, with symbol size proportional to the number of co-occurrences of restaurants and parking lots per quadnode.

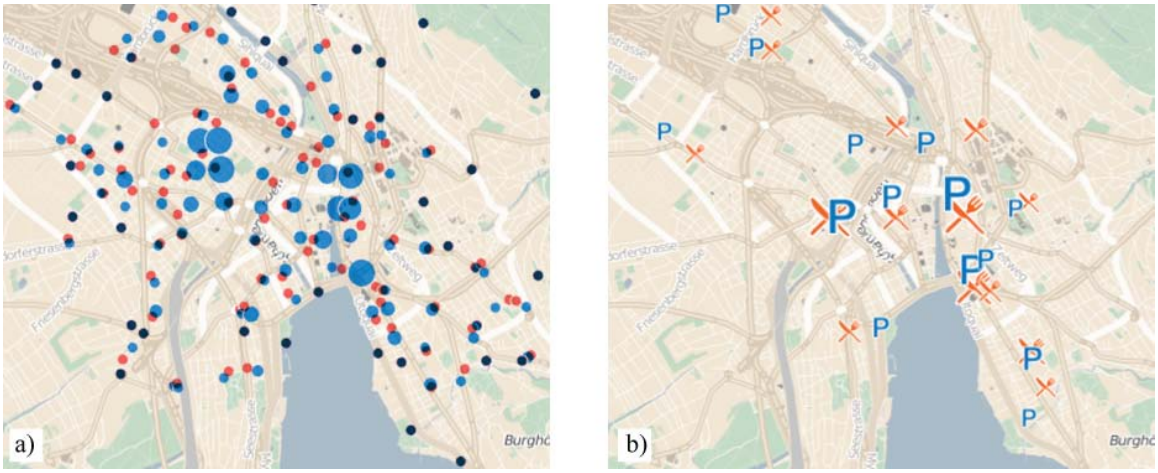


Figure 9 a) overlay of midpoint aggregation (cyan, with symbol size weighted by number of aggregated points), and rank-based selection (red) for LOD 13, overlaps shown in dark blue; b) aggregation by collocation, with symbol size weighted by the number of co-occurrences for restaurant and parking. Base map: © 2012 CloudMade – map data CC BY SA 2012 OpenStreetMap.org

To further resolve spatial conflicts in a map and/or retain more elements than for instance in the case of centrality-based simplification a displacement algorithm can be applied (Fig. 10). The displacement algorithm tries to accommodate as many points as possible, keeping at most one point per quadnode. Thus, it denotes the highest possible holding capacity for the current LOD given cartographic legibility constraints. On the other hand, it also homogenizes dense clusters and thus affects the overall distribution pattern (compare Fig. 9a and 9b). The proposed displacement algorithm only displaces a point to the neighbor of a quadnode if the neighboring quadnode has a holding capacity higher than one for the current LOD. If for a particular quadnode no displacement is needed, no check is applied for further potential overlaps with points contained in quadnodes adjacent to the node's boundary. Thus, some overlaps are still visible in Figure 10b, which could be resolved by an additional check while applying the displacement algorithm. However, the displacement result (Fig. 10b) is capable of displaying more points than the corresponding centrality-based simplification (Fig. 10a), since it was possible to remove

some overlaps and move points to adjacent quadnodes. In general, for all algorithms that do not maintain the original point positions (i.e. in aggregation, displacement operations), points may be moved to positions that violate possible constraints of the background map (e.g. restaurants may fall inside a lake). The maximum displacement distance of an aggregated or displaced point is half the diagonal of a quadnode at the target LOD and therefore linked to the selected symbol size. Thus, the maximum possible displacement can be controlled. Furthermore, including background constraints and adding a mask layer of positions to avoid (e.g. a waterbody mask for the restaurants) would remove that issue but also make the algorithm less generic.

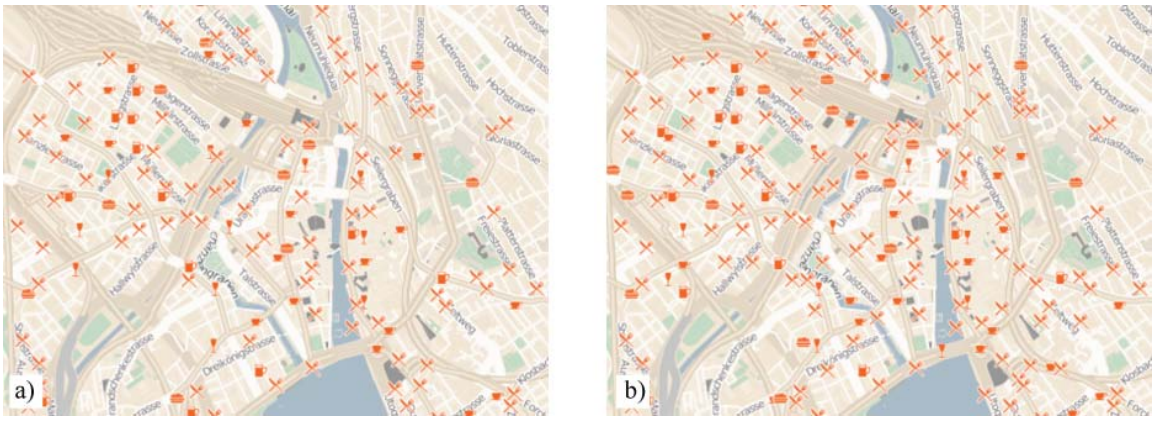


Figure 10 a) result of centrality-based simplification, displaying 138 elements; b) displacement applied after simplification based on centrality, displaying 184 elements. Base map: © 2012 CloudMade – map data CC BY SA 2012 OpenStreetMap.org

A brief performance test of the computation time used for quadtree creation as well as for selected generalization algorithms is shown in table 2, for three datasets of increasing volume. The computation time further depends on the spatial distribution of the dataset, the implementation, and the architecture used. The measure distinguishes between the average creation time of the quadtree for each dataset and the average time needed to update between the individual zoom levels, with no caching option applied. While there is no significant difference between the selection and aggregation algorithms, displacement with its neighborhood check is (not surprisingly) more costly. For very large datasets the caching option would further reduce the update time between the zoom levels and provide a smoother zooming experience to the user.

Table 2: Average execution time for three different datasets, for quadree creation and update time between two consecutive different zoom levels, on a dual-core Pentium CPU running at 3.16 GHz, with Windows 7 and Java 1.6.

	Small data set ~290 points	*Medium data set ~2800 points*	*Large data set ~86000 points*
Quadtree creation	0.8 ms	8 ms	460 ms
Computation time to move between two different zoom levels			
Selection	0.13 ms	3 ms	150 ms
Median Selection	0.06 ms	2 ms	110 ms
Aggregation	0.05 ms	1.6 ms	90 ms
Displacement	0.98 ms	13 ms	800 ms

Finally, Figure 11 illustrates in a series of steps how different generalization operations may be applied and chained together. As a reference the raw, ungeneralized input data is shown at LOD 14 in Figure 11a. In a first step (Fig. 11b) centrality-based simplification is used to relax spatial conflicts, while largely maintaining the spatial distribution of the input data. Remaining spatial conflicts are then further resolved by the displacement algorithm in (Fig. 11c). Bereuter and Weibel (2012) introduced 'content zooming', which provides the user with the capability to change the amount and granularity of foreground information presented, while keeping the geometric map scale the same. Content zooming is intended to allow overriding the effects of 'standard' map generalization, focusing on optimized content representation from the perspective of information seeking by a mobile user. In Figure 11 content zooming is shown by overriding standard settings for displacement: Figures 11d and 11e depict displacement results for an increased and a decreased LOD, respectively. Zooming out (Fig. 11d) and in (Fig. 11e) on the content enables the user to get a better idea of the spatial distribution and allows interactively adapting the amount of map content to his/her needs. Note also that while in Figures 11a to 11c, the length of the indicator bars for 'scale' and 'content zoom' is the same, the 'content zoom' bar becomes shorter or longer than the 'scale' bar, respectively, in Figures 11d and 11e. The number of points shown in Figures 11a to 11e are: 122, 50, 55, 18, 103.

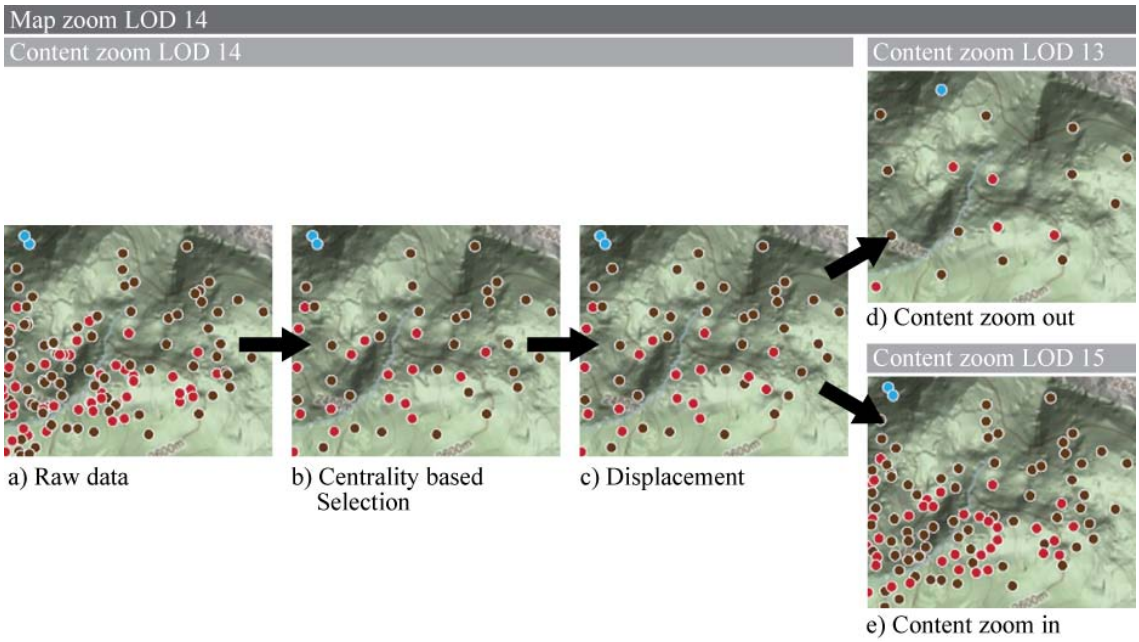


Figure 11 Series of steps of the application of different generalization algorithms (see text). Base map: © 2012 Google

How one selects and aggregates elements as representatives for the next scale, in order to abstract and reduce the information portrayed, has a strong influence on how the resulting pattern is perceived by the user. Due to the regular subdivision structure of the quadtree, quadtree-based generalization may lead to a certain degree of homogenization of the point distribution. However, even if strong homogenization occurs (as it occurs in the quadnode center aggregation algorithm), this effect can be alleviated by scaling the symbol size of the resulting points by the number of original points they represent, thus giving the user a better idea of the underlying spatial pattern (cf. Fig. 9).

Conclusions

On-the-fly generalization of point data is required to legibly display foreground data in online map services for web and mobile mapping, such as web map mashups and LBS. This paper makes several contributions that should help addressing this issue, which has been largely ignored by research and practice so far. We have shown that the quadtree offers a very useful framework to design generalization algorithms for point data that are capable of operating in real-time, including the construction of the PR quadtree. We have proposed a comprehensive set of quadtree-based algorithms that implement the major generalization operators on point data: selection, simplification, aggregation, and displacement. We have also shown how the quadtree data structure can additionally be used as a caching structure. The experiments conducted in this study demonstrate the application of the generalization algorithms, using two very different datasets of POI data and point collections, respectively. We have also shown an example of combining several different algorithms into workflow sequences, including 'content zooming' (Bereuter and Weibel, 2012).

In this paper, only a few experiments have been included, due to limited space. They may be sufficient to illustrate the potential of using quadtrees for point data generalization. More systematic experiments will be required to thoroughly assess the strengths and weaknesses of the proposed algorithms, and compare them to 'classical' generalization algorithms. Further evaluation should also include usability testing with users in real-life information seeking tasks. Several algorithmic improvements and extensions are possible, such as the inclusion of spatial constraints imposed by the background map (e.g. the lake and the street network in the POI example, or topographic barriers in the animal observations example), or more elaborate displacement strategies. Clearly, such extensions will have to be balanced against the added computational effort required, or they might become unfeasible for clients and thus call for server-side computing. Finally, the integration and interaction of different generalization operators and algorithms (such as in the last of the above examples) might also be studied further.

Acknowledgements

The research reported in this paper represents part of the PhD project of the first author. Funding by the Swiss National Science Foundation through project *Generalisation for Portrayal in Web and Wireless Mapping (GenW2+)* (SNF no. 200020-138109) is gratefully acknowledged.

References

Bereuter, P. and Weibel, R. (2010) Generalisation of point data for mobile devices: A problem-oriented approach. In: *Proc., 13th Workshop on Progress in Generalisation and Multiple Representation ICA Commission on Generalisation and Multiple Representation*.

Bereuter, P. and Weibel, R. (2011) A Diagnostic Toolbox for Assessing Point Data Generalisation Algorithms. In: *Pro., 25th Intl. Cartographic Conference*, Paris, 3-8 July 2011.

Bereuter, P., Weibel, R. and Burghardt, D. (2012) Content zooming and exploration for mobile maps, In: *Proc., 15th AGILE International Conference on Geographic Information Science*, pp. 74-80.

de Berg, M., Bose, P., Cheong, O., Morin, P., & de Berg, M. (2004) On simplifying dot maps. *Computational Geometry* 27(1): 43-62.

Burghardt, D., Purves, R. S. and Edwardes, A. J. (2004) Techniques for on the-fly generalisation of thematic point data using hierarchical data structures. In: *Proceedings, GIS Research UK 12th Annual Conference*.

Dutton, G. (1999a) A Hierarchical Coordinate System for Geoprocessing and Cartography. *Lecture Notes in Earth Science* 79. Berlin, Germany: Springer-Verlag.

Dutton, G. (1999b). Scale, sinuosity and point selection in digital line generalization. *Cartography and Geographic Information Science* . 26(1): 33-53.

Edwardes, A. J., Burghardt, D. and Weibel, R. (2005) Portrayal and Generalisation of Point Maps for Mobile Information Services. In: Meng, L., Zipf, A. & Reichenbacher, T. (eds.) *Map-based Mobile Services: Theories, Methods and Implementations*. 2, pp. 11-28. Berlin, Germany: Springer-Verlag.

Lehto, L. and Sarjakoski, L. T. (2005) Real-time generalization of XML-encoded spatial data for the Web and mobile devices. *International Journal of Geographical Information Science* 19(8-9): 957-973.

McMaster, R. B. and Shea, K. S. (1992) Generalization in Digital Cartography. Association of American Geographers, Washington D.C.

Moore, M. and Wilhelms, J. (1988) Collision detection and response for computer animation. *ACM SIGGRAPH Computer Graphics* 22(4): 289-298.

Reichenbacher, T., & De Sabbata, S. (2011). Geographic relevance: different notions of geographies and relevancies. *SIGSPATIAL Special*, *3*(2): 67-70.

Samet, H. (1989) *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Reading, MA, Addison-Wesley Longman Publishing Company.

Samet, H. (1990) *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison-Wesley Publishing Company.

Töpfer, F. and Pillewizer, W. (1966) The principles of selection. *Cartographic Journal*, 3(1):10-16.

van Oosterom, P. (2005) Variable-scale Topological Data Structures Suitable for Progressive Data Transfer: The GAP-face Tree and GAP-edge Forest. *Cartography and Geographic Information Science*, *32*(4): 331–346.

van Oosterom, P. and Meijers, M. (2011) Towards a true vario-scale structure supporting smooth-zoom. *14th ICA/ISPRSWorkshop on Generalisation and Multiple Representation & the ISPRS Commission II/2 WG on Multiscale Representation of Spatial Data*.

Weibel, R., and Burghardt, D. (2008) Generalization, On-the-Fly. In: Shekhar, S. & Xiong, H. (eds.) *Encyclopedia of GIS*. pp. 339–344, New York: Springer.